GEOMETRIC IDENTIFICATION OF OBJECTS IN CIVIL ENGINEERING APPLICATIONS

I. Bilchuk*

* *Moscow State University of Civil Engineering, Moscow, Russia*
E-mail: irina.bilchuk@gmx.net

Keywords: geometric identification, graphical user interface, method of Guttmann, object oriented R-trees, test bed.

Abstract. *Objects for civil engineering applications can be identified with their reference in memory, their alpha-numeric name or their geometric location. Particularly in graphic user interfaces, it is common to identify objects geometrically by selection with the mouse. As the number of geometric objects in a graphic user interface grows, it becomes increasingly more important to treat the basic operations add, search and remove for geometric objects with great efficiency.*

Guttmann has proposed the Region-Tree (R-tree) for geometric identification in an environment which uses pages on disc as data structure. Minimal bounding rectangles are used to structure the data in such a way that neighborhood relations can be described effectively. The literature shows that the parameters which influence the efficiency of the R-trees have been studied extensively, but without conclusive results.

The goal of the research which is reported in this paper is to determine reliably the parameters which significantly influence the efficiency of R-trees for geometric identification in technical drawings. In order to make this investigation conclusive, it must be performed with the best available software technology. Therefore an object-oriented software for the method is developed. This implementation is tested with technical drawings containing many thousands of geometric objects. These drawings are created automatically by a stochastic generator which is incorporated into a test bed consisting of an editor and a visualisor. This test bed is used to obtain statistics for the main factors which affect the efficiency of R-trees. The investigation shows that the following main factors which affect the efficiency can be identified reliably :

- *number of geometric objects on the drawing*
- *the minimum und maximum number of children of a node of the tree*
- *the maximum width and height of the minimal bounding rectangles of the geometric objects relative to the size of the drawing.*

1 INTRODUCTION

The concept of Region-Trees (R-trees) for the geometric identification of objects was first presented by Antonin Guttmann [1] in 1984. His central idea was to enclose each geometric shape by a rectangular box whose faces are orthogonal to the coordinate axis. The box becomes the placeholder for the shape, and all geometric operations are performed on the box. Since the box is rectangular and axis-oriented, these operations are simple and fast. They can be implemented in efficient procedures.

Consider a set of geometric shapes with associated boxes. Assume that one of the following types of search is to be performed on the set :

- find all shapes which contain a given point
- find all shapes which overlap a given shape.

In the approach of Guttmann, the search is first performed on the boxes which are associated with the shapes. This preliminary procedure significantly reduces the number of shapes which must be investigated in detail, since only those shapes must be considered whose boxes satisfy the search criterion. For shapes in a plane, the reduced set may consist of a set of polygons. Additional methods are then applied to test whether the polygons of the reduced set contain the given point or overlap the given polygon. These refined methods require the determination of the union and intersection of polygons and are highly sophisticated if special cases such as touching and self-touching of polygons are included.

The method of Guttmann and variations of his method have been studied intensively [2,3,4]. The influence of parameters such as the number of shapes, the sequence of entry and removal of shapes, the size limits for the nodes of the R-tree and the aspect ratios of the boxes on the efficiency of the method have been investigated. The results of the measurements reported in the literature for the basic operations :

- add an object to the R-tree
- remove an object from the R-tree
- search for overlap of a test box with boxes of the R-tree

show large ratios of the standard deviation to the average value of the required time and do not permit reliable conclusions.

The R-tree implementation of Guttmann is based on pages as primary data structure. The aim of the research reported in this paper is to implement the method in an object-oriented environment and to develop statistic methods which permit reliable conclusions concerning the primary parameters which influence the efficiency of the method.

2 REGION - TREES

Since this paper treats R-trees in the context of technical drawings, the properties of R-trees for geometric objects in a plane are considered. The drawing consists of a set of shapes which are bounded by polygons and which may contain openings with a polygonal boundary. The properties of these shapes are defined, read and computed by a set of methods which form an interface Shape in Java. The R-tree is a structured set of Shape objects, which is decomposed into two subsets :

- Objects which are the contents of the tree: these objects are called user shapes. They are instances of classes of the user application which implement the Shape interface, for instance the class Polygon for geometric objects whose outline consists of closed polygons.
- Objects which describe the structure of the tree: these objects are instances of class Node. The user of the R-tree is not concerned with the nodes of the tree. The nodes are used to manage the contents of the tree.

The addition and removal of user shapes in an R-tree and the search for user shapes relative to a test point or by overlap with a test shape can follow each other in arbitrary sequence.

The basic concept for R-trees is the minimal bounding rectangle (box), denoted by MBR and shown in Fig. 1. The MBR of a single shape is a rectangle whose edges are parallel to the axes of the coordinate system of the drawing. The coordinates of the edges are chosen so that all points of the shape are contained in the MBR, and that each edge of the box has at least one point in common with the shape.

Boxes are topologically closed. Two boxes are equal if their location and size parameters x, y, w, h are equal. The union of two boxes is the box which tightly encloses both boxes. The intersection of two boxes consists of the points which are contained in both boxes : it can be empty or consist of a single point, an edge or a box. Two boxes overlap if they have at least one common point.

If a box is intersected with another box and the result is empty, the state of this box is set to empty. The union of a box x with an empty box is x. The intersection of a box with an empty box is an empty box.

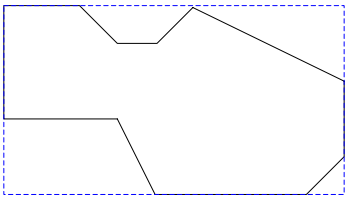

Figure 1 : Geometric shape with minimal bounding rectangle (box)

The MBR of a set of shapes is a rectangle which contains all points of all shapes of the set, as shown in Fig. 2. Each edge of the MBR contains at least one point of a shape of the set.

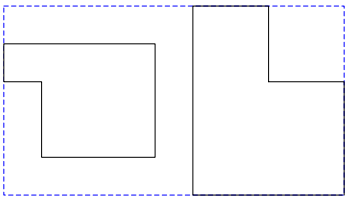

Figure 2 : Minimal bounding rectangle for a set of shapes

Each node of the tree knows its own name, the name of its parent node and the names of its child nodes, as shown in Fig. 3. The data structure which holds the names of the children of a node is called the container of the node. The sequence of the children in the container of a node is arbitrary. In addition, the MBR of the children of a node is contained in the node. This

MBR is called the box of the node. The boxes of nodes change as the structure of the tree changes.

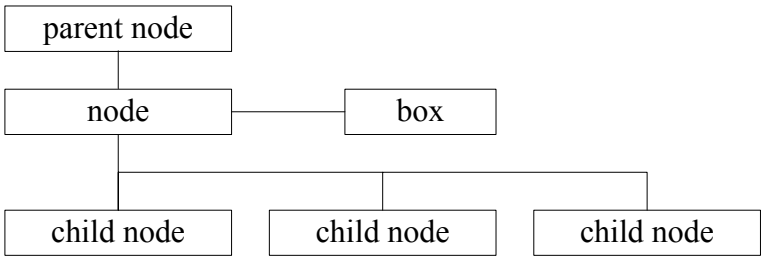

Figure 3 : Node structure of the R-tree

In contrast to a node, a user shape, as shown in Fig. 4, does not have children. Its box contains a single shape (the user shape itself) and is constant. The user shape can contain information on its specific geometric shape, such as coordinates for bounding polygons.

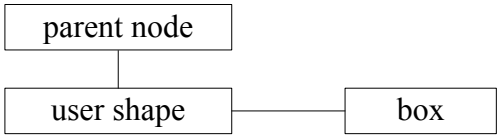

Figure 4 : User shape of the R-tree

In the data structure for the R-tree the empty tree consists of a root node and a stem node. The name of the root node is by convention "_root_", and the name of the reference of the root node in the source code is by convention "root". The root node has a single child, the stem node, as shown in Fig. 5. It contains the name of the stem node, which is converted into the address a2 of the body of the stem node by a call to the data base. In an empty tree, the stem node does not have any children.

As the height of a tree changes, different nodes serve as stem nodes, but the root node remains the same independent of the contents of the tree. The current stem node is identified by the property that the name of its parent node is "_root_".

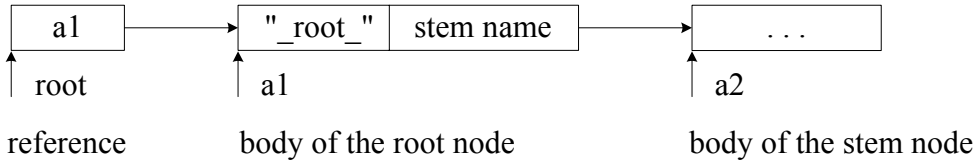

Figure 5 : Root and stem nodes of an R-tree

As user shapes are added to the tree, their names are added to the container of the stem node. The maximum number N of children of a node is a specified parameter of the R-tree. When this number is reached for the stem node, its shape set is split, as shown in Fig. 6, and assigned to two nodes which are children of a new stem node. The algorithm for the split is described in section 3. The stem is the only node for which the minimum container size cannot be specified, but must be 2 if it contains nodes and 0 if it contains user shapes (empty tree).

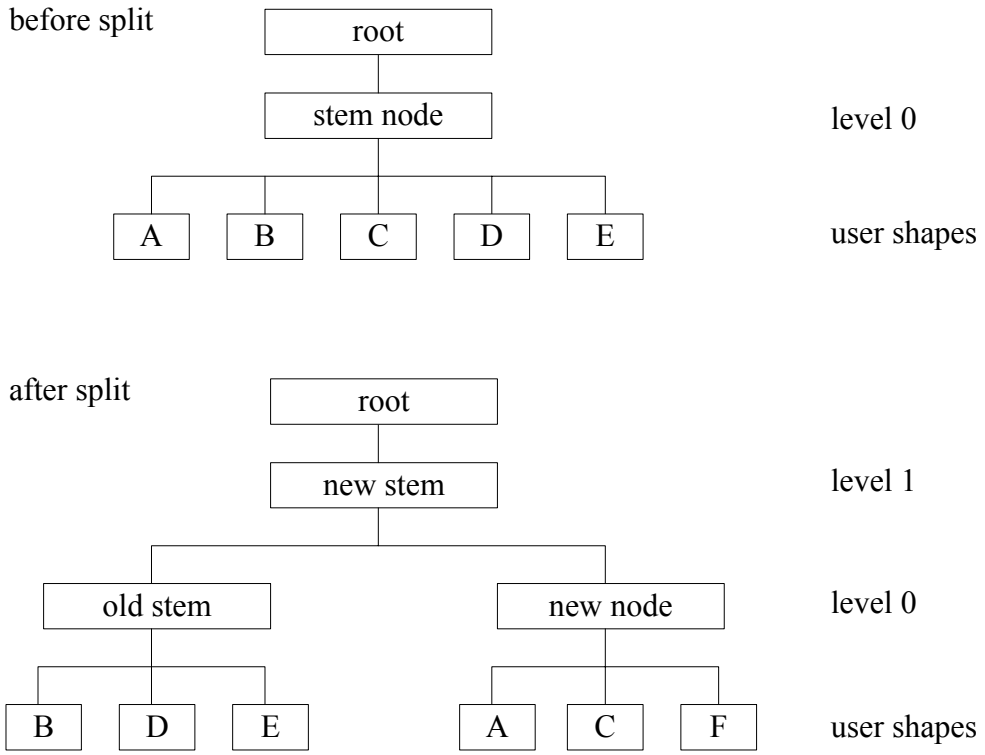

Figure 6 : Node split in an R-tree

The figure shows that R-trees are fully balanced trees. The user shapes lie in a common level. The nodes which are the parents of user shapes form the level 0 of the tree, their parents form the level 1, and so on. The stem node, which was in level 0 before the split, remains in level 0 after the split but is no longer the stem node.

If an additional user shape is added to the tree, the node A at level 0 which is most suitable as container for the shape is determined. If node A already contains the maximum number of shapes, its shape set is split. The subsets are assigned to node A and a new node B. The split is propagated along the path from node A to the root and terminates if a node container is found which does not overflow. As the structure of the tree changes, the attributes of the boxes of its nodes are recomputed.

In order to remove a user shape, the node at level 0 which contains the shape is determined. The shape is removed from the container of this node. If this reduces the container size below the minimum specified for the nodes of the tree, or below 2 in the stem node, the node is removed from the tree by removing its name in the container of its parent. This condensation of the tree is propagated towards the stem and terminates if a node container is found which does not underflow. The stem node is treated as a special case, since the number of levels of the tree is reduced after underflow of the stem node. The size of the node boxes is adjusted to the change in the structure of the tree.

The nodes which are removed from the tree are the roots of subtrees which contain active nodes and user shapes. These are reinserted into the tree at the level from which they were removed, so that all user shapes and nodes retain their level in the tree. The algorithm is similar to the algorithm for shape addition to the tree.

3 NODE MANAGEMENT

A node is a container for an unstructured set consisting of at least MIN and at most MAX shapes which are the children of the node. The stem node forms an exception: its set contains at least 2 nodes (or no user shapes if the tree is emty) and at most MAX shapes.

If an attempt is made to add a child to a node which is full, the node is split. The rules for the split operation are not unique and have a significant influence on the performance of the R-tree. The rules which are implemented here are the original rules of Guttman from 1984, which could not be improved significantly by subsequent research.

The split of the child set in the container of the node into two subsets is started by considering all unordered child pairs x, y from the container of this node. Each pair x, y is enclosed tightly with a box. The increase in area from the sum of the areas of the boxes for x and y to the area of the enclosing box is computed. The pair which leads to the largest increase in area is chosen as seeds so that set1 = {x} and set2 = {y}. By this choice, the largest possible increase in box area is avoided.

If the number of remaining children in the container of the node is the number required to give set1 or set2 the required minimum size, these children are added to that set and the algorithm terminates. Otherwise, the next child to be added to either set1 or set2 is picked so that of all remaining children in the container, it has the greatest preference for one of the two sets, as follows :

For each remaining child in the container, the difference in the increase of the area of the two boxes enclosing set1 and set2 is determined. The child with maximal difference is added to the set whose box area is increased least by the added child.

If a node A, which is not the stem, is split into nodes A and B, it is attempted to store B in the container of the parent of A. If this container is full, its child set is split. This process is continued until a node is encountered whose container is not yet full, or until the stem node is reached. If the stem node is split, a new stem node C is constructed so that the number of levels in the tree is increased by 1. The children A and B are stored in the container of node C.

If a child is removed from a node Z whose container has the minimum container size, node Z is removed from the container of its parent node. If this causes an underflow at the parent node, the process is repeated. If there is underflow at a stem node whose container holds nodes (i.e. not user shapes), there is exactly one node in the container which becomes the new stem of the tree: the number of levels in the tree is reduced by 1. If there is an underflow at a stem node which contains user shapes, the stem is retained. It then holds either one child, or it is empty. If the stem is empty, the tree as a whole is empty.

4 VARIATION OF R – TREE STRUCTURE

The number of shapes in an R-tree is large. The investigation has shown that the addition or removal of a single shape in the tree frequently leads to a substantial restructuring of the tree. It proved to be very difficult to follow the changes in the structure on the basis of alpha-numeric output of the tree parameters, particularly the node data. At the same time, it became evident that insight into the restructuring of the tree was essential for understanding of the method, and to provide a lead for possible enhancements.

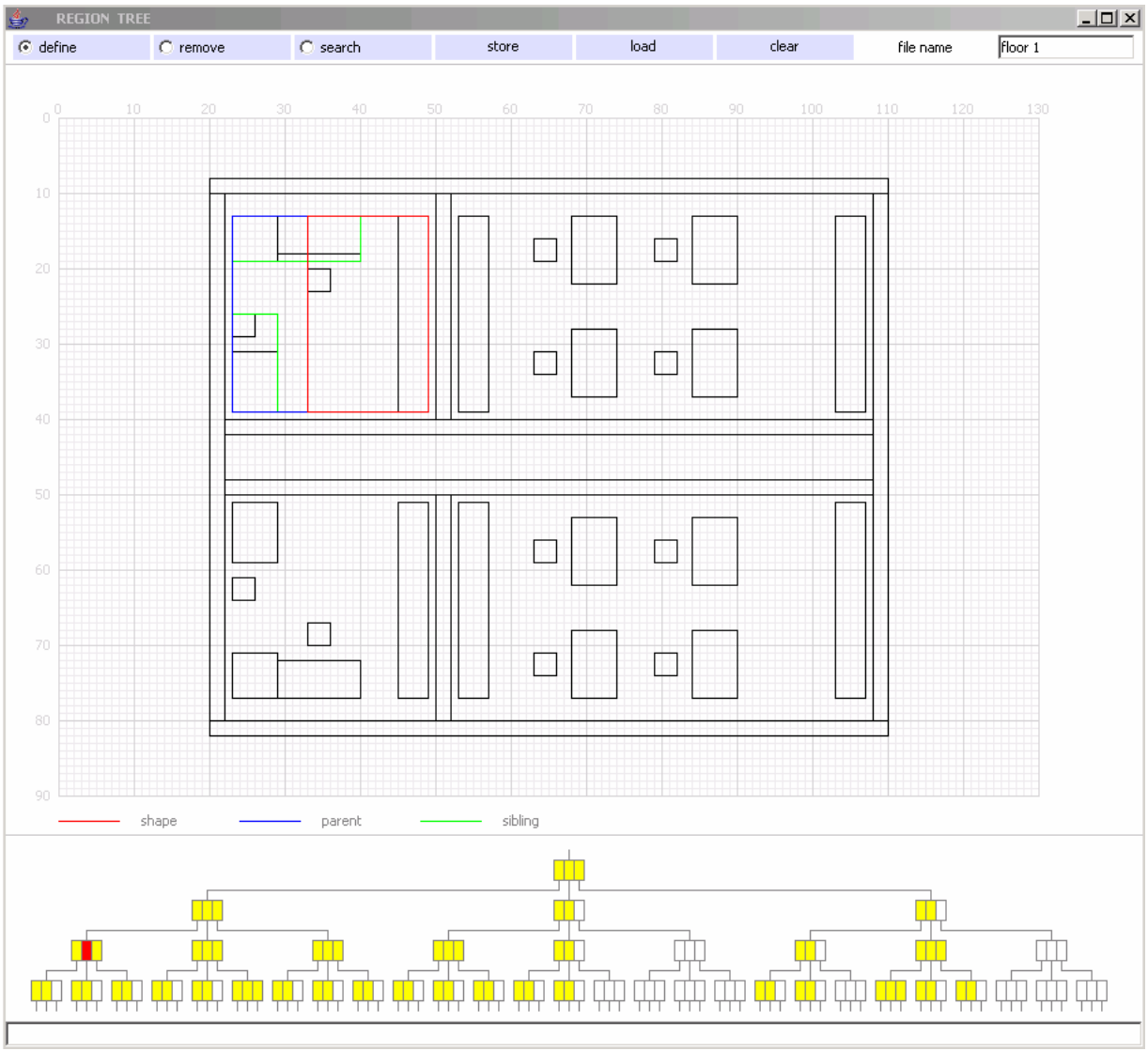

Figure 7 : Visual inspection of the R-tree structure

In order to evaluate the variations in the R-tree structure during add and remove operations, the interactive graphical interface of Fig. 7 was developed. The upper part of the window shows user shapes on a grid. The lower part of the window shows the nodes and references of the R-tree. The container of each node in this example contains at most 3 children. Empty slots in a container are shown in white, occupied slots in yellow color. One of the shapes is selected by mouse click in the lower part of the window. The slot of this

shape is marked red in the R-tree and the shape is marked red in the grid. The siblings of the selected shape are marked in the grid with green color.

The simple example with at most 81 shapes in the R-tree clearly demonstrated that the change in the structure of the tree due to the addition or removal of a shape cannot be anticipated reliably in an intuitive fashion. It was therefore decided to base all further research on statistical evaluation for a large number of shapes per drawing. This led to the development of the generator, editor and visualisor which are presented in the next sections of this paper.

5 R – TREE TEST GENERATOR

The generator performs tests with which the mean value and the standard deviation of the time required for the basic operations add, remove and search in R-trees are determined. Each performance test is concerned with exactly one of the types of operation on R-trees :

type 1 : add a polygon to the tree
type 2 : search for overlap of a test box with the polygons of the tree
type 3 : remove a polygon from the tree.

A performance test consists of measurements at a specified number of stations. Each measurement yields a mean value and a standard deviation for the duration of the operation for a specific set of polygons. All measurements are made for the same number of polygons, whose geometry, however, varies from station to station. The mean value and the standard deviation at a station are determined from a series of a specified number of observations on the polygon set of the station. The mean value and the standard deviation of a random variable x are computed with the usual formulas :

mean value $$m = \frac{1}{n}\sum_i x_i \qquad (1)$$

$$q = \sum_i \left(x_i - m\right)^2 \qquad (2)$$

deviation $$s = \sqrt{q/(n-1)} \qquad (3)$$

x_i - i-th random value of variable x
n - number of random values in the set

The nature of the observation in the series depends on the type of the tree operation.

A performance test "**ADD**" consists of the following steps :

1. Initialize all random number generators and construct the set of polygons.
2. Construct a permutation of the indices of the polygons.
3. Add the polygons to the tree in the order determined by the permutation in step 2. Measure the time required and find the mean time per addition.
4. Repeat steps 2 and 3 a specified number of times for different permutations of the indices to account for the influence of the sequence of the polygon addition on the duration of the add operation. Compute the mean value and the standard deviation for the set of

observations. These values are denoted by m[i] and s[i] and are considered to be the result at the station with index i in the performance test.

5. Repeat steps 1 to 4 for the specified number n of stations in the test. Compute the mean value and the standard deviation of the sets m[0], m[1], ..., m[n-1] and s[0], s[1], ..., s[n-1].

A performance test "**SEARCH**" consists of the following steps :

1. Initialize all random number generators and construct the set of polygons.
2. Add the polygons to the tree.
3. Construct a specified number of search boxes and store the boxes in an array.
4. Search the tree for all of the boxes in the array. Measure the time required and find the mean time per search.
5. Repeat steps 2 to 4 a specified number of times to account for the influence of the set of search boxes and of the tree structure on the duration of the search operation. Compute the mean value and the standard deviation for the set of observations. These values are denoted by m[i] and s[i] and are considered to be the result at station i of the performance test.
6. Repeat steps 1 to 5 for the specified number n of stations in the test. Compute the mean value and the standard deviation of the sets m[0], m[1], ..., m[n-1] and s[0], s[1], ..., s[n-1].

A performance test "**REMOVE**" consists of the following steps :

1. Initialize all random number generators and construct the set of polygons.
2. Add the polygons to the tree.
3. Construct a random set of indices for the specified number of polygons which are to be removed from the tree.
4. Remove the polygons. Measure the time required and find the mean time per removal.
5. Repeat steps 2 to 4 a specified number of times to account for the influence of the set and sequence of the removed polygons and of the tree structure on the duration of the remove operation. Compute the mean value and the standard deviation for the set of observations. These values are denoted by m[i] and s[i] and are considered to be the result at station i of the performance test.
6. Repeat steps 1 to 5 for a specified number n of stations in the test. Compute the mean value and the standard deviation of the sets m[0], m[1], ..., m[n-1] and s[0], s[1], ..., s[n-1].

The numbers of observations per series and of stations per test are chosen so that the test result is not changed significantly if their value is increased. They are therefore internal to the test procedure.

The result of a performance test is a function of the following parameters :

- type of the investigated tree operation
- number of polygons in the tree
- minimum number of children of a node of the tree
- maximum number of children of a node of the tree
- minimum box width of a polygon
- minimum box height of a polygon

- maximum box width as a fraction of the width of the drawing
- maximum box height as a fraction of the height of the drawing
- width of the drawing which contains the user shapes
- height of the drawing which contains the user shapes

The performance test is repeated for R-trees which have different values of these properties. These values are set in the editor of the test bed. The results of the performance test are displayed in the visualisor, as shown in section 7 of this paper.

The random construction of the polygons in the tree and of the test boxes for searches, as well as the selection of the indices for the add and remove operations, are not part of the algorithms for the R-tree. The time required for the polygon construction and for the index selection are therefore not included in the measured times.

The construction of a polygon or a search test box consists of the following operations :

- generate random coordinates for the midpoint of the box
- generate random numbers for the width and height of the box
- eliminate boxes which do not have the minimum dimensions and adjust boxes which do not lie completely inside the drawing
- construct the polygon.

The selection of an index in array polygon[] consists of the following operations :

- generate a double number x between 0.0 and 1.0
- multiply the number of polygons in the tree with x and cast the result into an integer value
- assure by means of an index list that the computed index has not been selected in a previous cycle of the selection procedure.

6 EDITOR OF THE TEST BED

The performance tests for R-trees and their presentation in the visualisor depend on a large number of parameters. These parameters are controlled with the editor shown in Fig. 8.

The editor has a triple function :

- It is used to set the parameters for the test calculations with the generator for R-trees, to trigger the computations and to show the mean value of the operation time as well as the value of its standard deviation for a single test run.
- It is used to collect the results for a set of tests which are to be presented in a diagram, to set the graphics context of this diagram and to trigger the display of the diagram.
- It is used to store the data for a set of diagrams in files and to manage a list of these files. Any diagram of the list can be displayed during a session. The file management is delegated to the file editor, which is called from this editor.

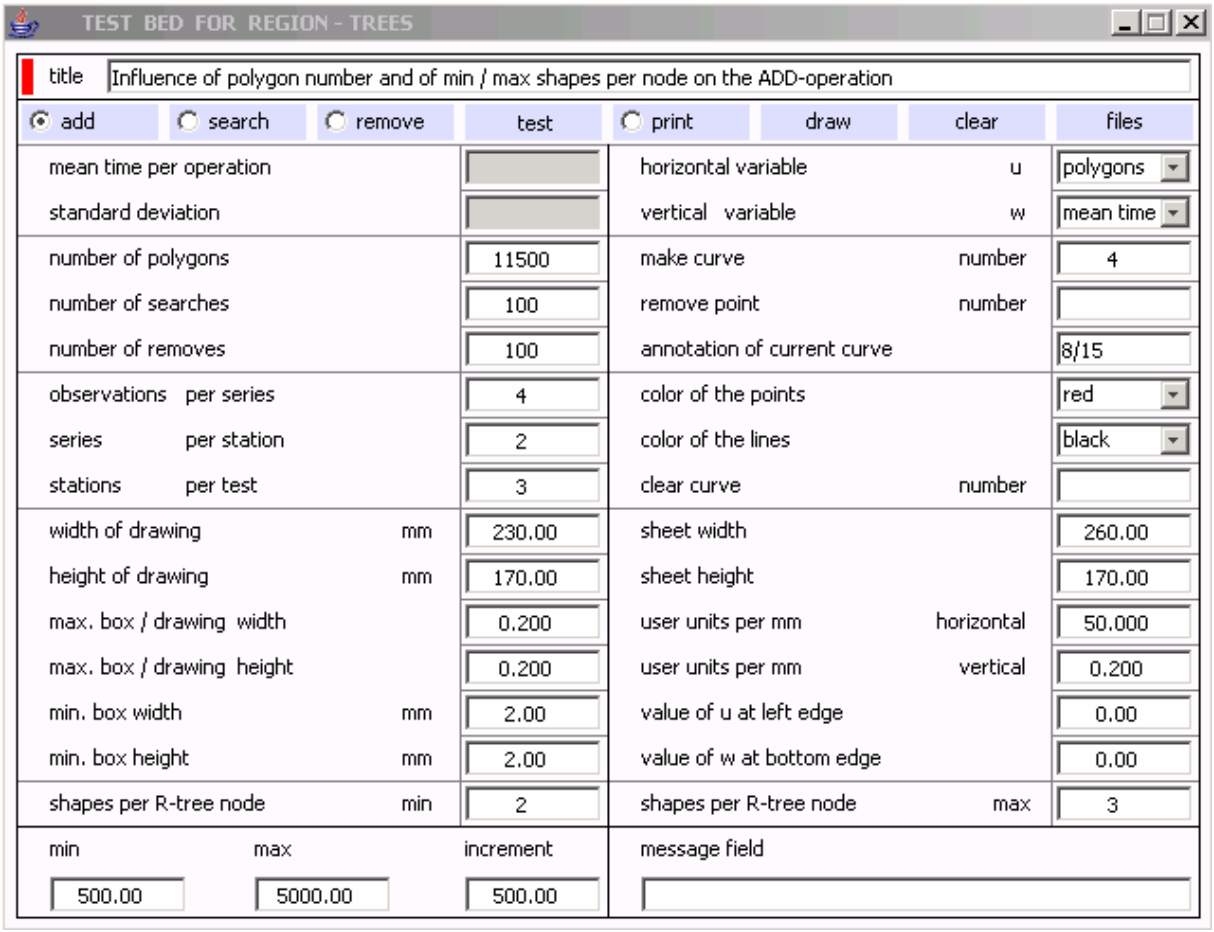

Figure 8 : Editor of the test bed

The top line of the editor is used to enter a title which is shown as heading in the diagram. It is intended to explain the contents of the diagram. At the left edge of the top line, a red bar is shown while the generator is computing the test result. During this time, the editor is blocked against input.

The bottom lines of the editor is used to set the parameters of the R-tree structure (minimum and maximum number of children per node) and to control loops for test series.

The left side of the editor panel is used to handle the tests with the generator. The radio buttons "add", "search" and "remove" form a button group and are used to select the type of R-tree operation that is to be tested. The button "test" is used to trigger the computations by the generator after the test parameters have been set in the lower text fields.

The right side of the editor panel is used to construct the diagram. If the radio button "print" is on, the generator prints intermediate results on System.out, which can be assigned to a file in the shell. If the button "draw" is clicked, the current state of the diagram is displayed. After this display is closed, the construction of the diagram can be continued. If the button "clear" is pressed and if this action is confirmed with key F1, all data of the diagram are disposed and cannot be recovered. The button "files" is used to call the file editor.

The variables for the horizontal and vertical axes of the diagram are assigned with combo boxes at the top of the right side of the editor panel. They apply to all curves in the diagram.

The diagram consists of a millimeter grid on which curves are drawn as polygons. The parameters of the grid are specified with the text fields in the lower part of the panel.

Each curve of a diagram carries a number in the range from 0 to 19. The curves need not be numbered consecutively. The points of a curve are numbered consecutively, starting with 0. The maximum number of points in a curve is 200. Each point of a curve corresponds to one test run of the generator. Each curve has the following attributes, which are set with the text fields and combo boxes at midheight of the panel:

- number of the curve
- color in which the points of the curve are painted
- color in which the lines of the curve are painted
- annotation, which is painted as string to the right of the last point of the curve.

The construction of a curve is started by entering the curve number. The points of the curve are computed consecutively with parameters of the left part of the editor panel. The result of each test run is automatically entered as a point in the curve. If the number of an existing curve is entered in the text field, the construction of this curve is continued with the successor of the previous end point.

The points and lines of a curve are painted with the color which is shown in the combo boxes when the curve is started. The colors for the current curve can be changed by resetting the combo boxes. If the construction of a curve is continued, the color combo boxes are automatically set to the colors which were previously selected for this curve.

The curve number is not shown in the diagram. The annotation of the curves is set by the user in the annotation text field. When the first point of a curve is stored, the contents of the annotation text field is automatically stored as annotation of the curve. The curve annotation can be changed by entering the new annotation in the text field. If the construction of a curve is continued, the annotation which was previously stored for this curve is automatically displayed in the text field.

A point of the current curve can be removed by entering its number in the point text field. The numbers of the succeeding points of the curve are reduced by one. A curve is removed from the diagram by entering its number in the clear text field. The data of the curve cannot be recovered.

7 RESULTS OF THE INVESTIGATION

The results of the investigation of the properties of R-trees, which were performed with the R-tree implementation, the generator and the editor described in the previous sections, are shown in the following diagrams which were prepared with the visualisor of the test bed that was developed in the project.

Figure 9 shows the variation of the mean time per add operation as a function of the number of polygons on the drawing for different values of the minimal and maximal size of the container of a node, as indicated by the marking k/m of the curves. It can be seen that the time required per add operation grows linearly with the number of polygons (shapes) on the drawing. The container size 2/3 should not be used for the add operation.

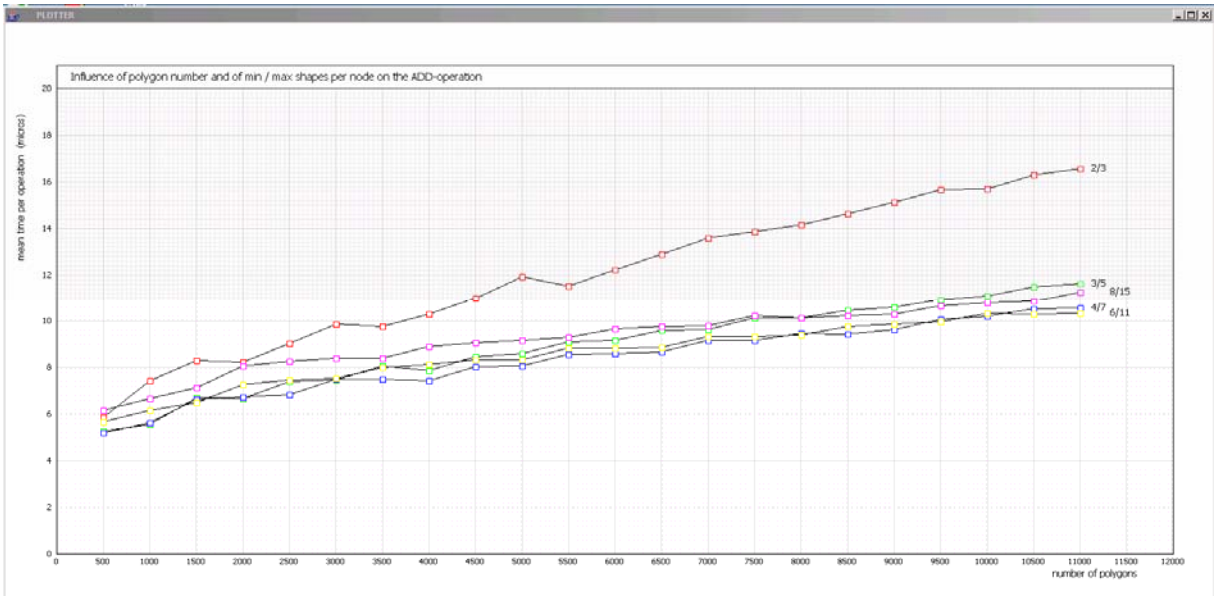

Figure 9 : Influence of the number of polygons on the mean time for the add operation

Figure 10 shows the variation of the mean time per remove operation as a function of the number of polygons on the drawing for different values of the minimal and maximal size of the container of a node, as indicated by the marking k/m of the curves. It can be seen that the time required per remove operation grows linearly with the number of polygons (shapes) on the drawing. The container sizes 2/3 and 3/5 should not be used for the remove operation.

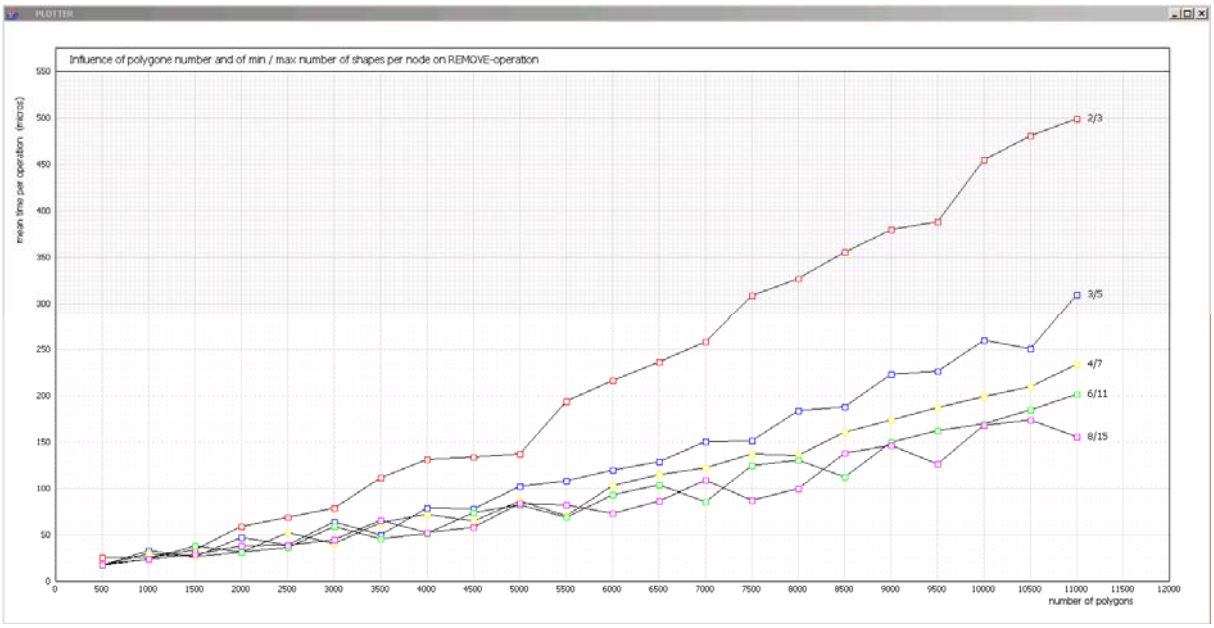

Figure 10 : Influence of the number of polygons on the mean time for the remove operation

Figure 11 shows the variation of the mean time per search operation as a function of the number of polygons on the drawing for different values of the minimal and maximal size of the container of a node, as indicated by the marking k/m of the curves. It can be seen that the time required per search operation grows nonlinearly with the number of polygons (shapes) on the drawing. The container sizes 2/3, 3/5 and 4/7 should not be used for the search operation.

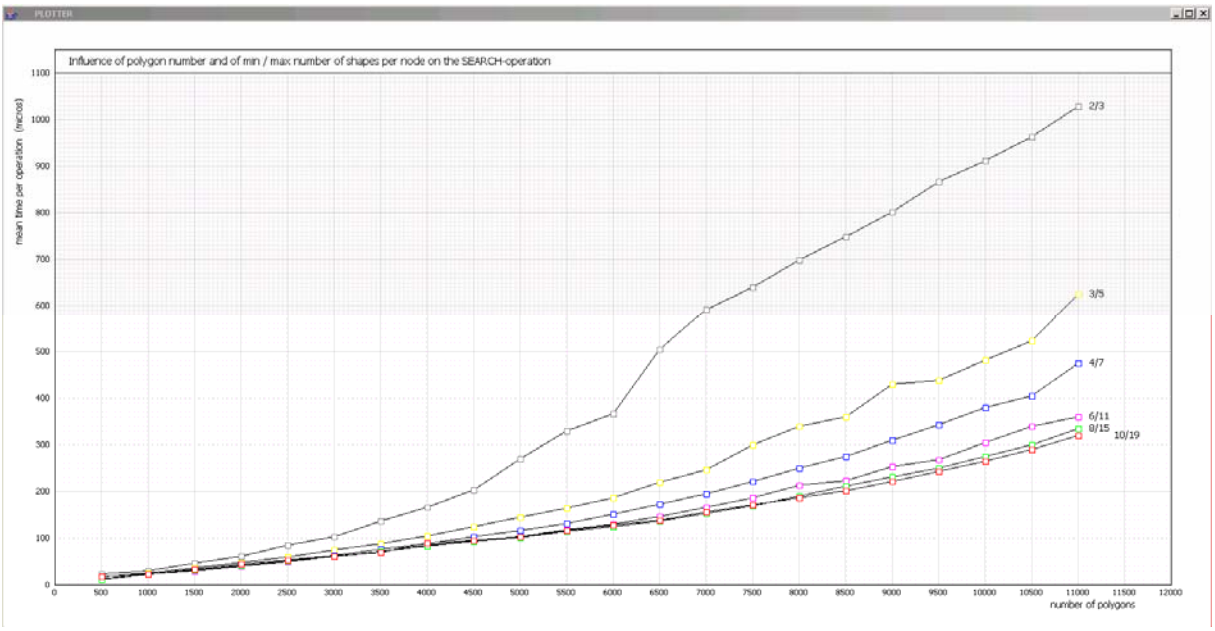

Figure 11 : Influence of the number of polygons on the mean time for the search operation

Figure 12 shows the variation of the mean time per add operation as a function of the maximum size of the container of a node for different values of the minimum size of the container of a node, as indicated by the marking of the curves. It can be seen that an increase of the minimum size of the container of a node up to the value 10 improves the efficiency of the add operation. For optimal minimum node size of the tree, the optimal maximum container size of a node lies between 10 and 30 children, depending on the minimal container size.

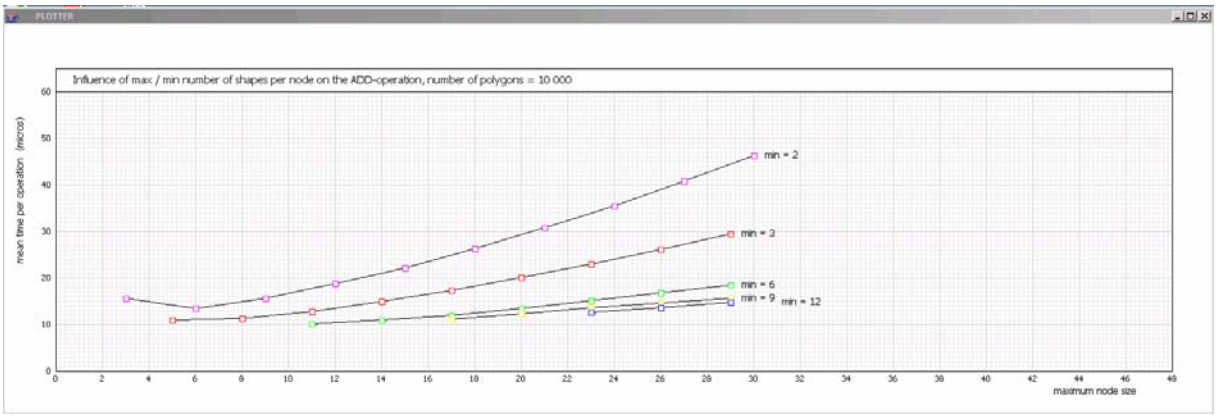

Figure 12 : Influence of the container size on the mean time for the add operation, number of polygons = 10 000

Figure 13 shows the variation of the mean time per remove operation as a function of the maximum size of the container of a node for different values of the minimum size of the container of a node, as indicated by the marking of the curves. It can be seen that an increase of the minimum size of the container of a node up to the value 10 improves the efficiency of the remove operation. For optimal minimum node size of the tree, the optimal maximum container size of a node lies between 10 and 30 children, depending on the minimal container size.

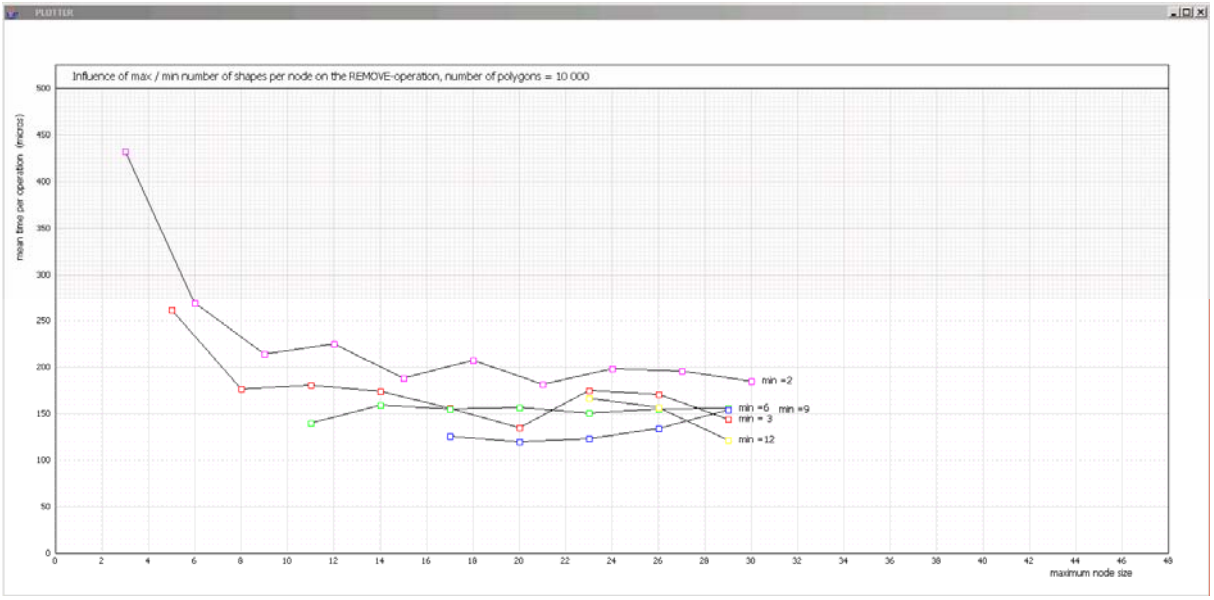

Figure 13 : Influence of the container size on the mean time for the remove operation, number of polygons = 10 000

Figure 14 shows the variation of the mean time per search operation as a function of the maximum size of the container of a node for different values of the minimum size of the container of a node, as indicated by the marking of the curves. It can be seen that an increase of the minimum size of the container of a node up to the value 10 improves the efficiency of the search operation. For optimal minimum node size of the tree, the optimal maximum container size of a node lies between 10 and 30 children, depending on the minimal container size.

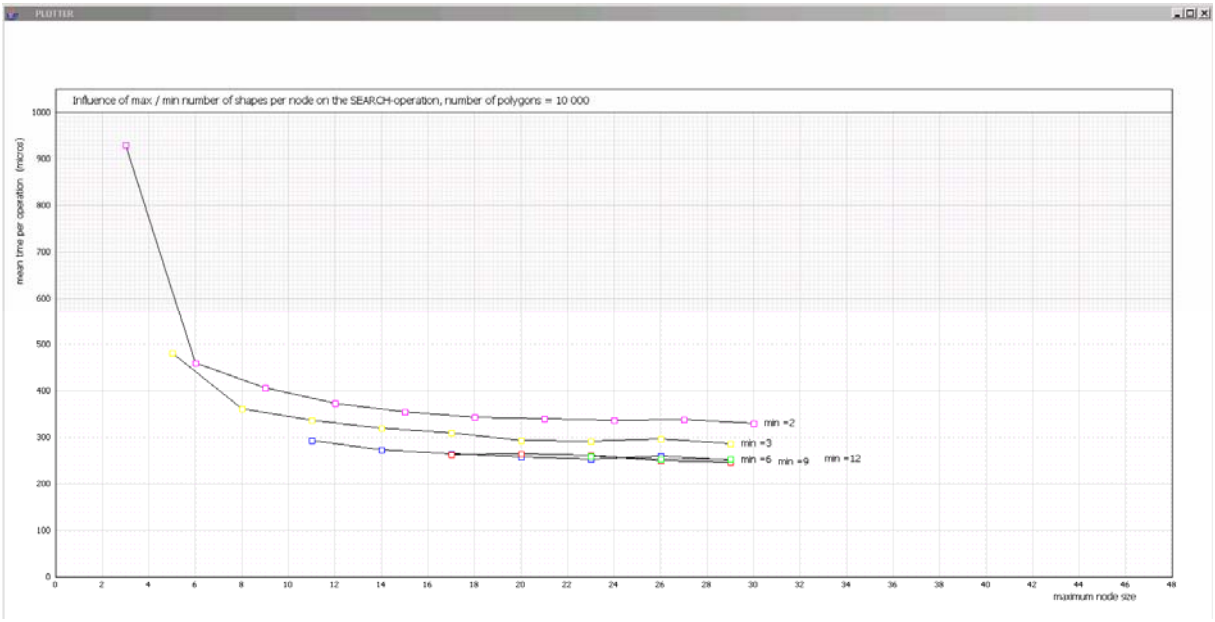

Figure 14 : Influence of the container size on the mean time for the search operation, number of polygons = 10 000

Figure 15 shows the variation of the mean time per add operation as a function of the maximum box height, expressed as a fraction of the height of the drawing, for different values of the maximum box width, expressed as a fraction of the width of the drawing, as indicated by the marking of the curves. It can be seen that the mean time per add operation is independent of the maximum box height and the maximum box width, expressed as a fraction of the drawing.

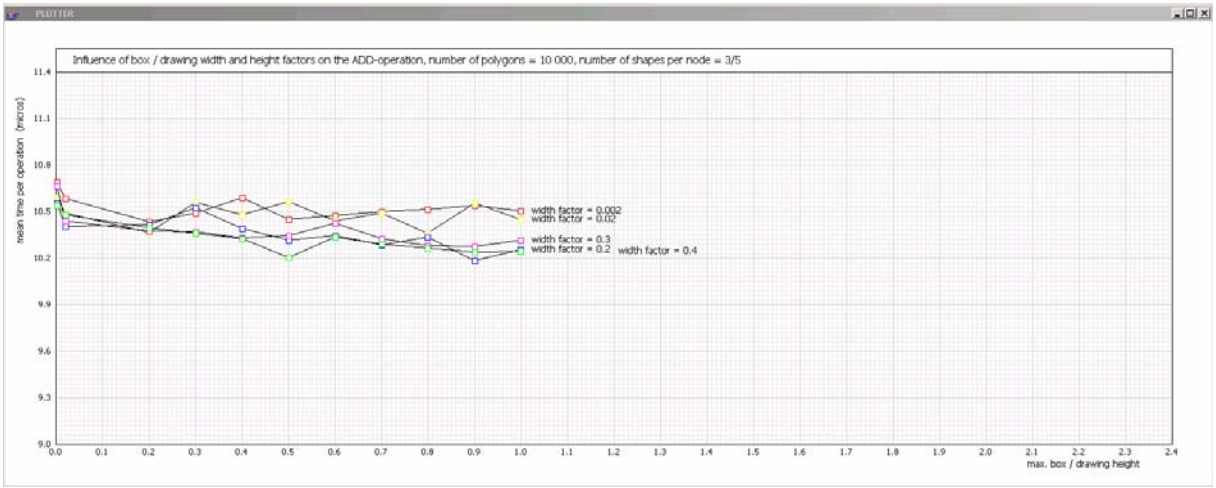

Figure 15 : Influence of the maximum box size on the mean time for the add operation, number of polygons = 10 000, container size = 3/5

Figure 16 shows the variation of the mean time per remove operation as a function of the maximum box height, expressed as a fraction of the height of the drawing, for different values of the maximum box width, expressed as a fraction of the width of the drawing, as indicated by the marking of the curves. It can be seen that the mean time per remove operation increases as the maximum box height and width increase.

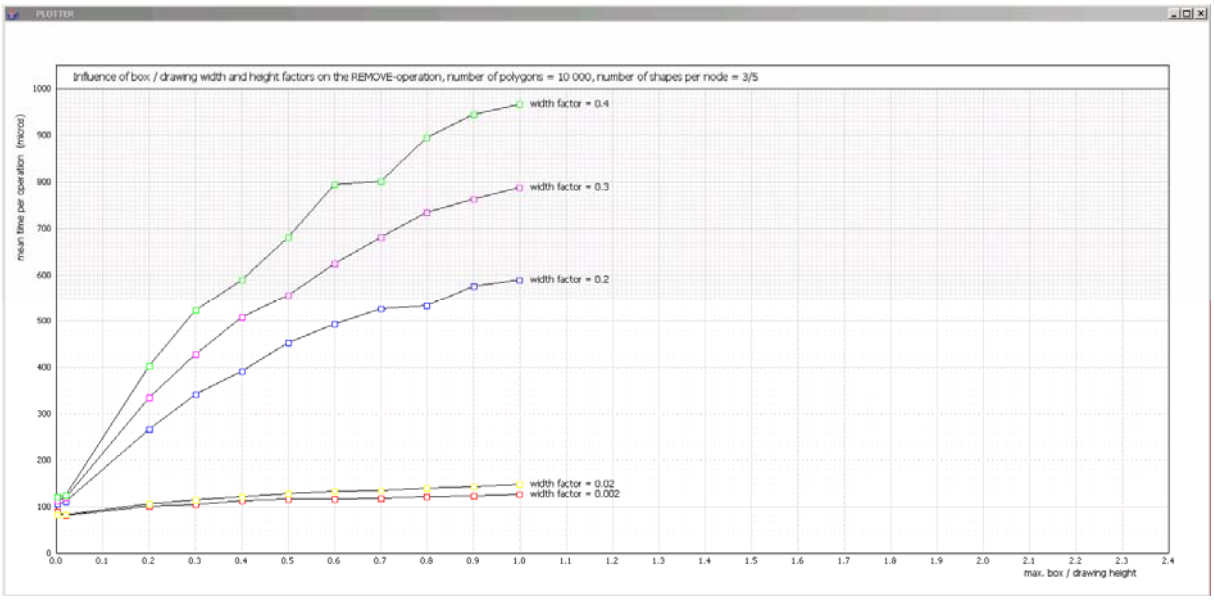

Figure 16 : Influence of the maximum box size on the mean time for the remove operation, number of polygons = 10 000, container size = 3/5

Figure 17 shows the variation of the mean time per search operation as a function of the maximum box height, expressed as a fraction of the height of the drawing, for different values of the maximum box width, expressed as a fraction of the width of the drawing, as indicated by the marking of the curves. It can be seen that the mean time per search operation increases as the maximum box height and width increase.

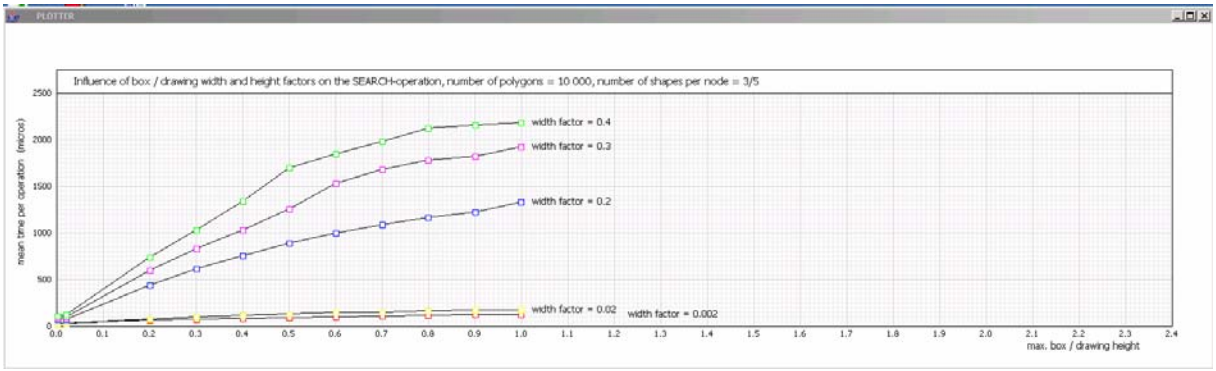

Figure 17 : Influence of the maximum box size on the mean time for the search operation, number of polygons = 10 000, container size = 3/5

Figure 18 shows the variation of the mean time per add operation as a function of the minimum box height for different values of the minimum box width, as indicated by the marking of the curves. It can be seen that the mean time per add operation is independent of the minimum box height and the minimum box width.

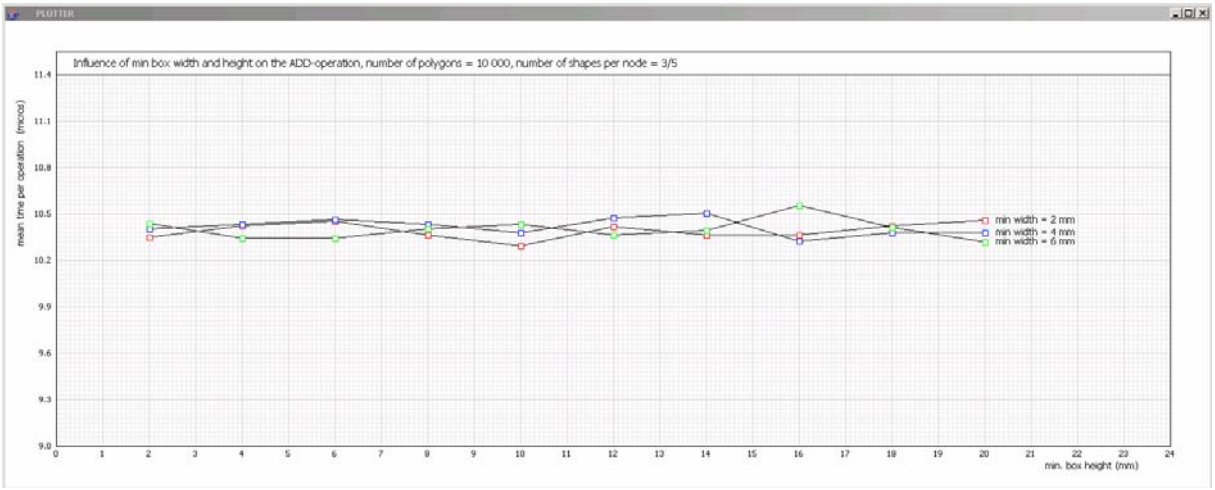

Figure 18 : Influence of the minimum box size on the mean time for the add operation, number of polygons = 10 000, container size = 3/5

Figure 19 shows the variation of the mean time per remove operation as a function of the minimum box height for different values of the minimum box width, as indicated by the marking of the curves. It can be seen that the mean time per remove operation is nearly independent of the minimum box height and the minimum box width.

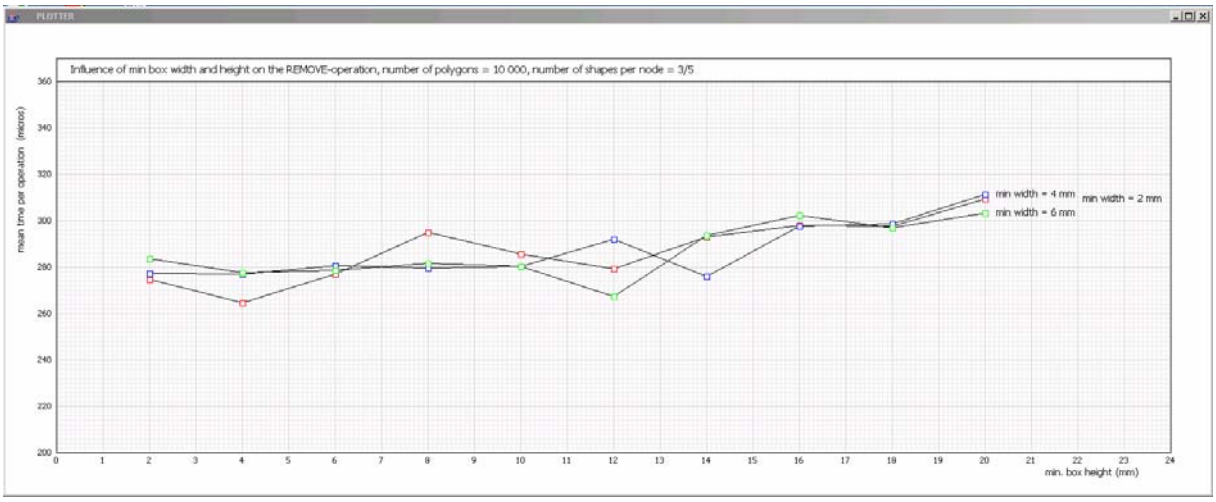

Figure 19 : Influence of the minimum box size on the mean time for the remove operation, number of polygons = 10 000, container size = 3/5

Figure 20 shows the variation of the mean time per search operation as a function of the minimum box height for different values of the minimum box width, as indicated by the marking of the curves. It can be seen that the mean time per search operation is nearly independent of the minimum box height and the minimum box width.

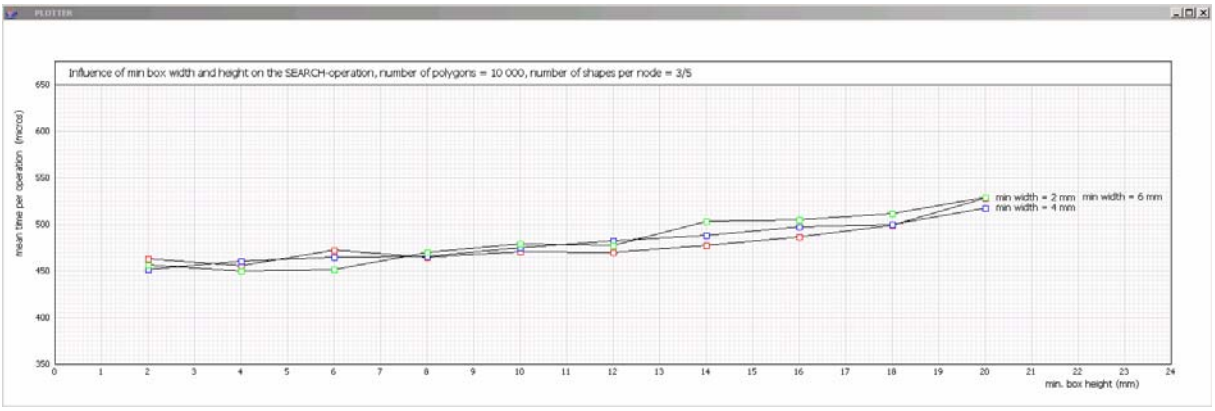

Figure 20 : Influence of the minimum box size on the mean time for the search operation, number of polygons = 10 000, container size = 3/5

Figure 21 shows the variation of the mean time for each of the basic operations as a function of the number of test stations. It can be seen that the mean time for each of the basic operations is independent of the number of test stations. Although the geometry of test shapes varies on the test stations, it does not affected the duration of basic operations.

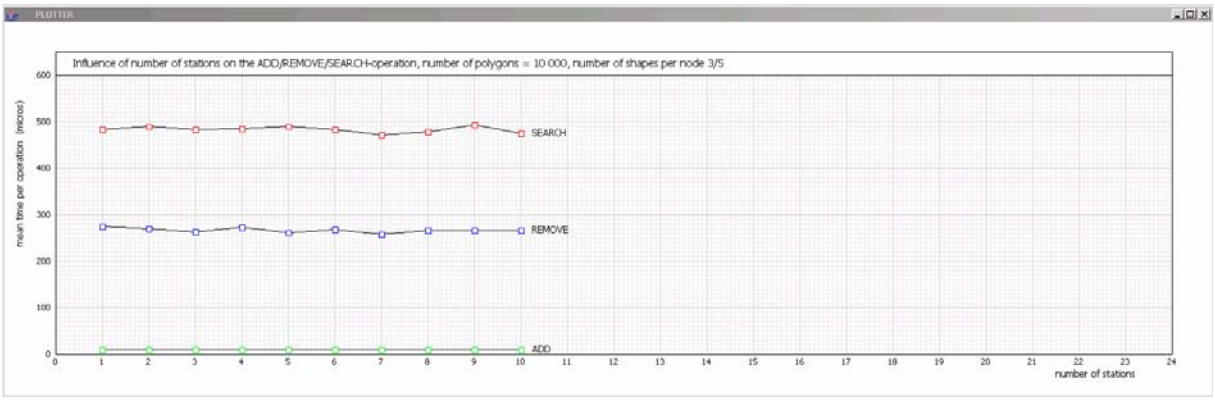

Figure 21 : Influence of the number of test stations on the mean time for basic operations, number of polygons = 10 000, container size = 3/5

Figure 22 shows the variation of the mean time for each of the basic operations as a function of the number of test observations. It can be seen that the mean time for each of the basic operations is independent of the number of test observations. Although the sequence of entry of the test shapes to the tree varies between observations, this does not affected the duration of the add operation. Although the geometry of the search boxes varies between observations, this does not affected the duration of the search operation. Although the test tree varies between observations, this does not affected the duration of the remove operation.

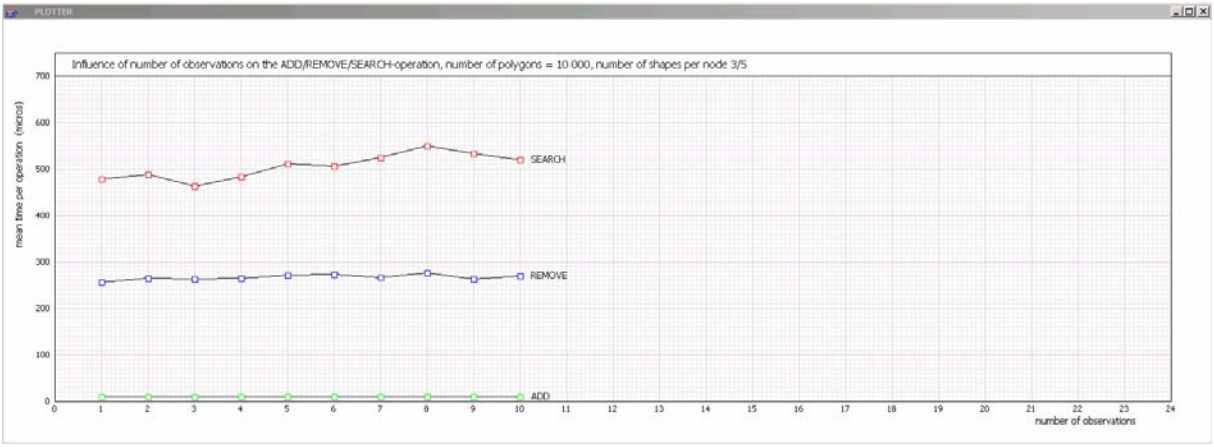

Figure 22 : Influence of the number of test observations on the mean time for basic operations, number of polygons = 10 000, container size = 3/5

8 CONCLUSIONS

The investigation permits the following conclusions for drawings with 10 000 shapes :

- An increase of the minimum size of the container in a node up to the value 10 improves the efficiency of the method for all basic operations.
- The optimal maximum container size of a node lies between 10 and 30 children.
- For optimal node size of the tree, the time required for a search operation exceeds the time required for an add operation by a factor 30.
- The time required for the remove operation is approximately half the time required for the search operation.
- For suitable node sizes of the tree, the time required per add or remove operation grows linearly with the number of shapes on the drawing.
- The time required for the search operation grows nonlinearly with the number of shapes on the drawing, but the nonlinearity is weak for suitable node sizes.
- As the maximum width and height of the shapes relative to the size of the drawing increases, the duration of the basic operations search and remove increases, whereas the duration is not affected for the add operation.
- The duration of the basic operations add, remove and search is not affected by the value of the minimum width and height of the shapes.
- All basic operations on R-trees are performed on a typical laptop in a time that is one order of magnitude faster than the human reaction time.

It is intended to continue the research in three areas :

- A study of the influence of the aspect ratio of the boxes on the performance of the R-tree.
- Development of a method to construct R-trees efficiently if all shapes to be entered in the tree are known at the outset.
- Development of alternatives to the R-tree concept.

The goal of the research project has been reached. The results show that the behavior of R-trees under different conditions can be evaluated reliably with the test bed that is presented in this paper.

9 ACKNOWLEDGEMENT

The research which is described in this paper was initiated in the research project DFG-GZ: 436 RUS 113/836/0-1 “Topology and Geometry of Cell-Oriented Building Modells” of the Deutsche Forschungsgemeinschaft (German Research Foundation DFG). It was continued in the research project RNP.2.1.2.9025 “Investigation of Mathematical and Computational Foundations for a Hybrid Method of the Geometric Identification of Objects in Graphical Information Sets of Civil Engineering” of the Ministry for the Education and Science of the Russian Federation. The author wishes to thank the foundations for their support.

The work on the first project was conducted in collaboration with Prof. Oleg Ignatiev, Prof. Vera Galishnikova of the Volgograd State University of Architecture and Civil Engineering and Prof. Peter Jan Pahl of the Technische Universität Berlin.

REFERENCES

[1] A. Guttman, R-Trees : A dynamic index structure for spatial searching. *In Beatrice Yormark, editor, SIGMOD`84, Proceedings of Annual Meeting,* Boston, Massachusetts, June 18-21, 1984, pages 47-57. ACM Press, 1984.

[2] O. Günther, Efficient Structures for Geometric Data Management. *Springer-Verlag* Berlin Heidelberg, 1988.

[3] M. van Kreveld, J. Nievergelt, T. Roos, P. Widmayer, Algorithmic Foundations of Geographic Information Systems. *Springer-Verlag* Berlin Heidelberg, 1997.

[4] Y. Manolopoulos, A. Nanopoulos, A.N. Papadopoulos, Y. Theodoridis, R-trees: Theory and Applications. *Springer-Verlag* London Limited, 2006.